



# Safety-Critical Systems: Rapprochement Between Formal Methods and Control Theory



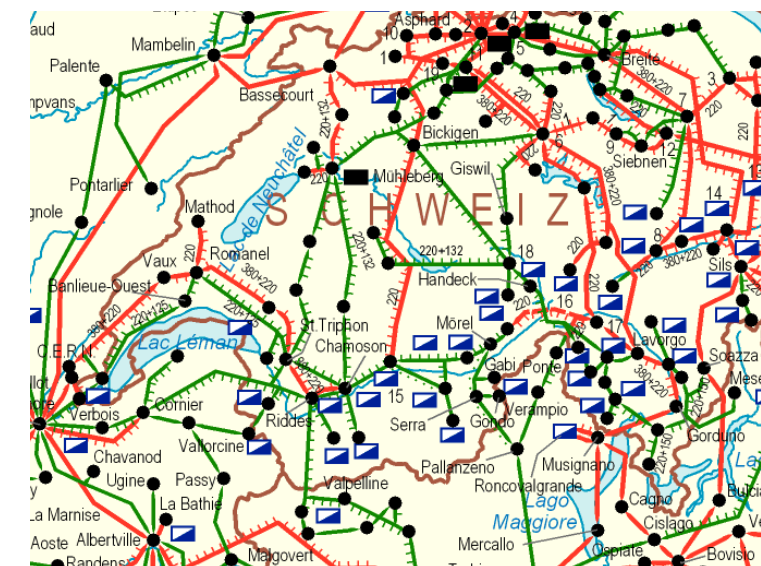
Richard M. Murray  
California Institute of Technology  
NASA Formal Methods  
9 May 2019



0.07 deaths every  $10^9$  miles

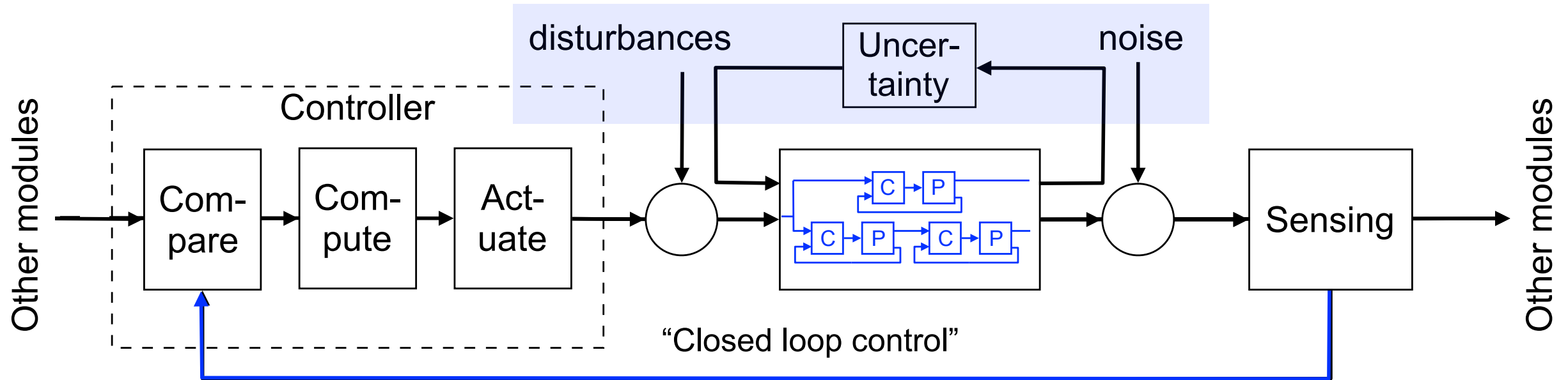
←  
??

7 deaths every  $10^9$  miles  
35K/year (US)



I. Savage, "Comparing the fatality risks in United States transportation across modes and over time", *Research in Transportation Economics*, 43:9-22, 2013.

# Control System “Standard Model”



## Key elements

- Process: input/output system w/ dynamics
- Actuation: mechanism for manipulating process
- Sensing: mechanism for detecting process state
- Compute: compare actual / desired; determine action
- **Environment: description of the uncertainty present in the system (bounded set of inputs/behaviors)**

## Advantages of feedback

- Design of dynamics
- Robustness to uncertainty
- **Modularity and interoperability**

## Disadvantages of feedback

- Increased complexity
- Potential for instability
- Amplification of noise

# Important Trends in Control in the Last 15 Years

## (Online) Optimization-based control

- Increased use of online optimization (MPC/RHC)
- Use knowledge of (current) constraints & environment to allow performance and adaptability

## Layering, architectures, networked control systems

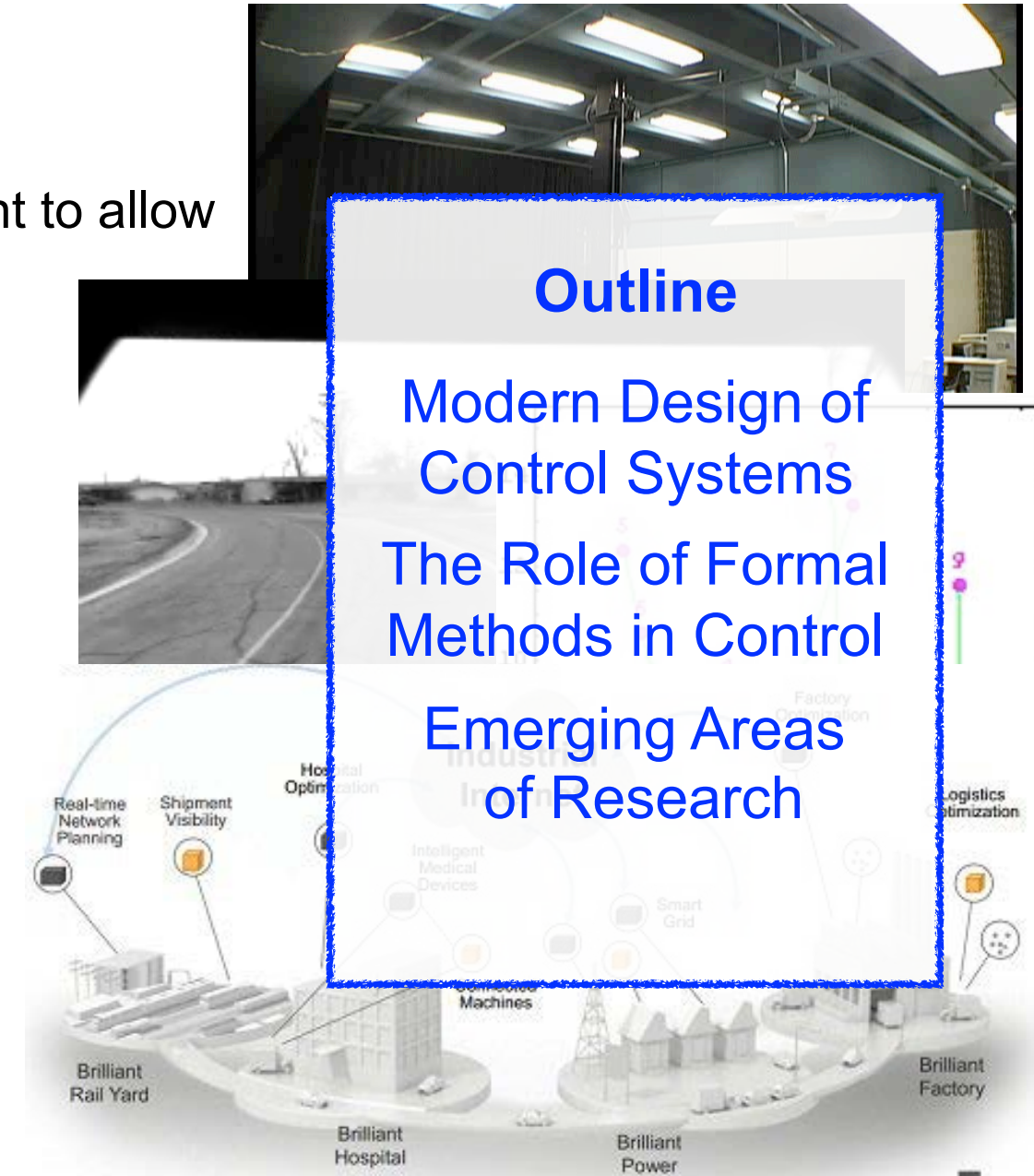
- Command & control at multiple levels of abstraction
- Modularity in product families via layers

## Formal methods for analysis, design and synthesis

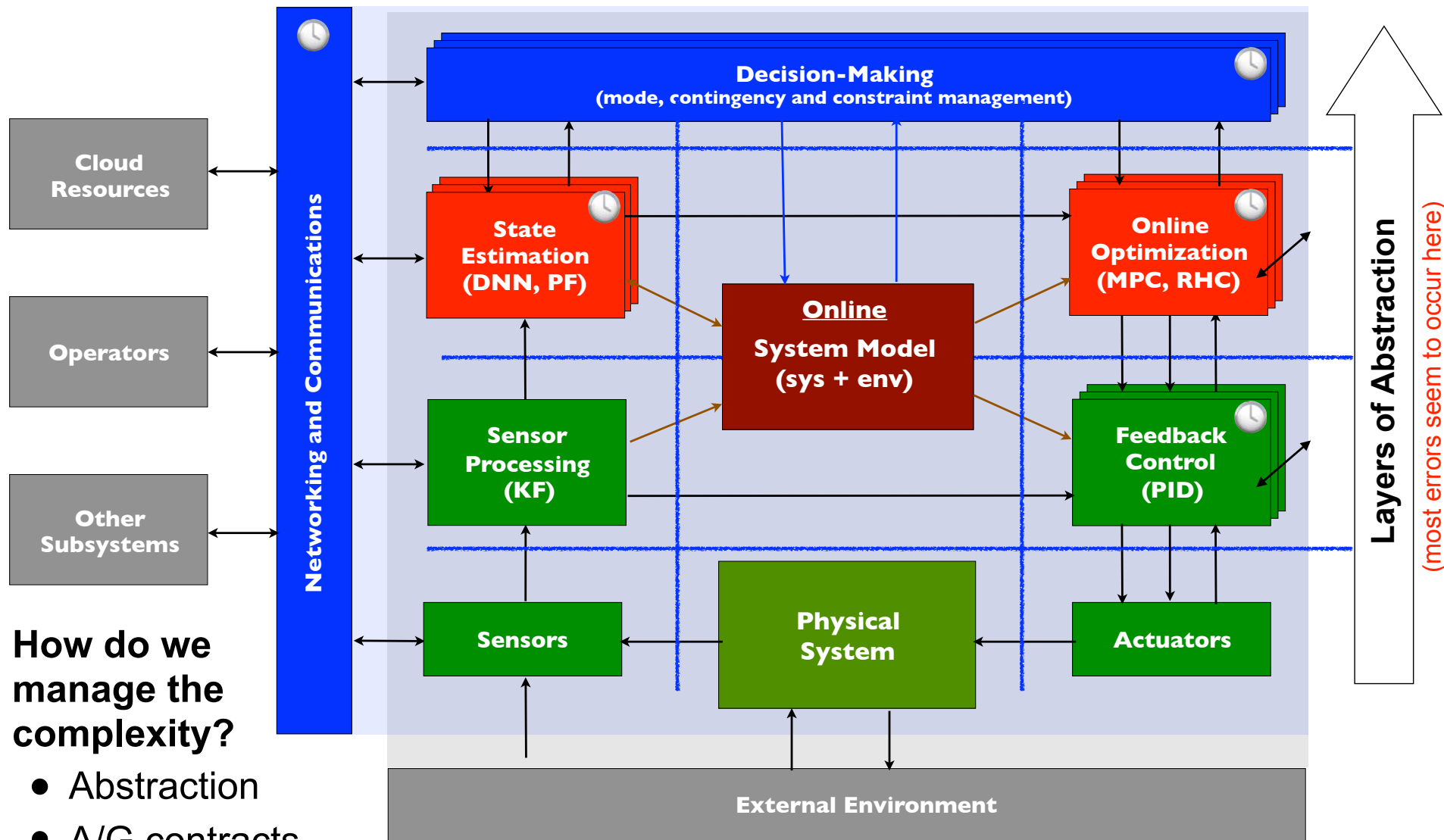
- Build on work in hybrid and discrete event systems
- Formal methods from computer science, adapted for “cyberphysical” (computing + control) systems

## Components → Systems → Enterprise

- Increased scale: supply chains, smart grid, IoT
- Use of modeling, analysis and synthesis techniques at all levels. Integration of “software” with “controls”



# Design of Modern (Networked) Control Systems



## How do we manage the complexity?

- Abstraction
- A/G contracts
- Formal methods for verification/synthesis + model- & data-driven sims/testing

## Examples

- Aerospace systems
- Self-driving cars
- Factory automation/ process control
- Smart buildings, grid, transportation

## Challenges

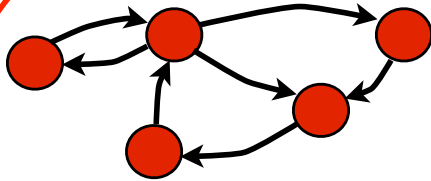
- How do we define the layers/interfaces (vertical contracts)
- How do we scale to *many* devices (horizontal contracts)
- Stability, robustness, security, privacy



# Abstractions Hierarchy for Networked Control Systems

Continuous:  $\dot{x} = f_\alpha(x, u, d)$   $\min J = \int_0^T L(x, u, \alpha) dt + V(x(T))$

Discrete:  $g(x, \alpha) \implies \alpha' = r(x, \alpha)$  if X then Y, never Z, always W, ...

	Level	Model	Specification
<b>Supervisory Control (FSM)</b>	Decision-Making		$(\phi_{\text{init}} \wedge \Box \phi_{\text{env}}) \implies (\Box \phi_{\text{safe}} \wedge \Box \Diamond_{\leq T} \phi_{\text{live}})$
<b>Online Optimization (RHC)</b>	Trajectory	$\dot{x} = f_\alpha(x, u)$ $g_\alpha(x, u, z) \leq 0$	$\min J = \int_0^T L_\alpha(x, u) dt + V(x(T))$
<b>Feedback Control (PID)</b>	Tracking	$y = P_{yu}(s) u + P_{yd}(s) d$ $\ W(s)d(s)\  \leq 1$	$\ W_1 S + W_2 T\ _\infty < \gamma$
<b>System Dynamics (ODE)</b>	Process	$\dot{x}^i = f_\alpha(x^i, u^i, d^i)$ $x \in \mathcal{X}, u \in \mathcal{U}, d \in \mathcal{D}$	Operating Envelope Energy Efficiency Actuator Authority

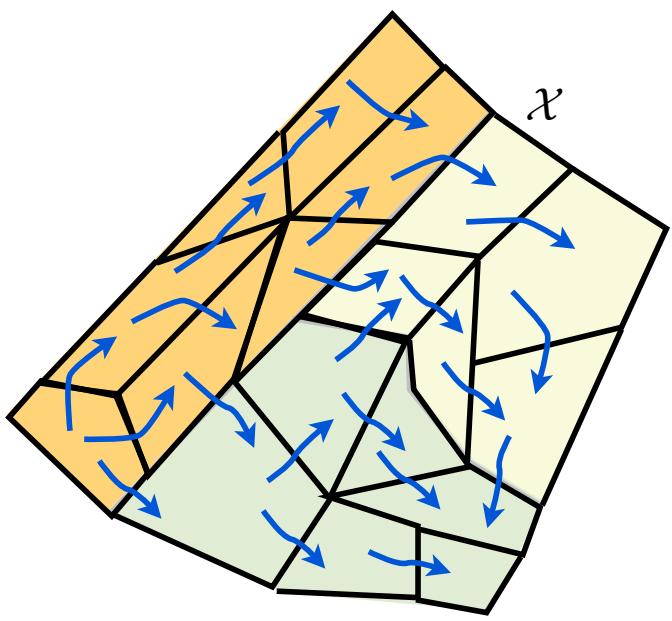
## Common approach

- Combine “layers” & solve “hybrid” design problem
- Example: MLD, SMT

## Preferred approach

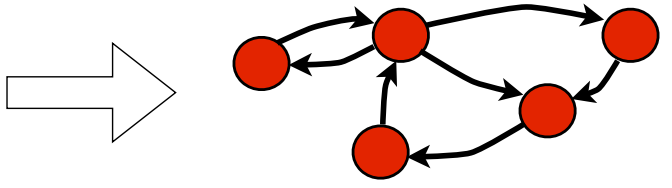
- Maintain separation of layers!
- Create *contracts* between layers:
  - simplified representation of other layers
  - explicit assume/guarantee structure

# Discrete Abstractions for (Hybrid) Dynamical Systems



Continuous states  $\rightarrow$  discrete abstractions

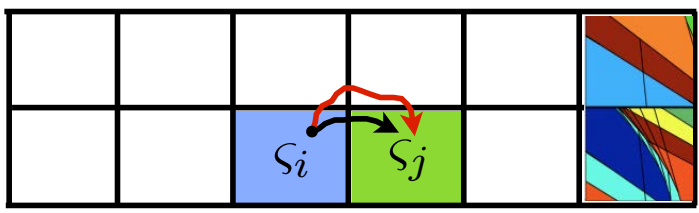
$$\dot{x} = f_{\alpha}(x, u)$$
$$g_{\alpha}(x, u, z) \leq 0$$



Use formal tools to create abstractions

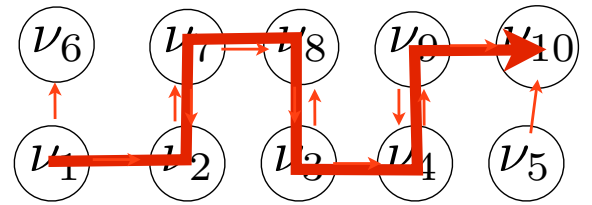
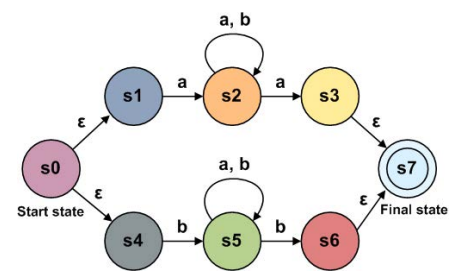
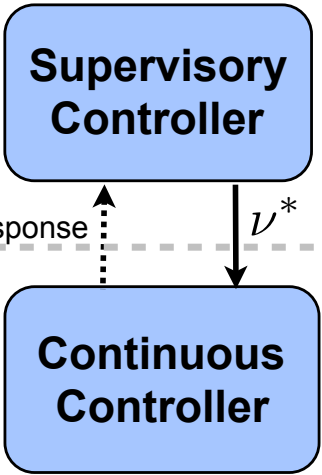
- Use reachability analysis (trajectory gener'n) to compute regions, transitions
- Account for disturbances, uncertainty, failures (using, for example, MPT)

- Look for regions such that we can move from one region to another w/out leaving the union of two regions

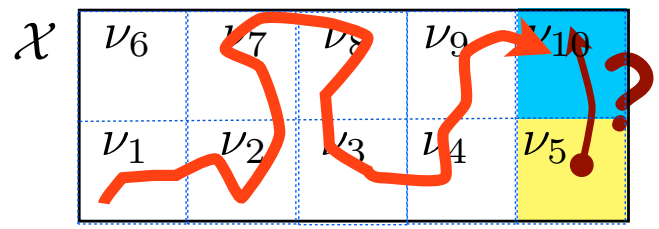


- Solve via trajectory generation algorithm: piecewise linear dynamics w/ disturbances:

$$s[t + 1] = As[t] + Bu[t] + Ed[t]$$
$$s[t] \in \varsigma_i, s[N] \in \varsigma_j, u[t] \in U$$



$$\min J = \int_0^T L_{\alpha}(x, u) dt + V(x(T))$$



$$L \begin{bmatrix} s[0] \\ u[0] \\ \vdots \\ u[N-1] \end{bmatrix} \leq M - G \begin{bmatrix} d[0] \\ \vdots \\ d[N-1] \end{bmatrix}$$

# Synthesis of Reactive (Feedback) Controllers

## Reactive Protocol Synthesis

- Find control action that insures that specification is always satisfied
- For LTL, complexity is doubly exponential (!) in the size of system specification

## GR(1) synthesis for reactive protocols

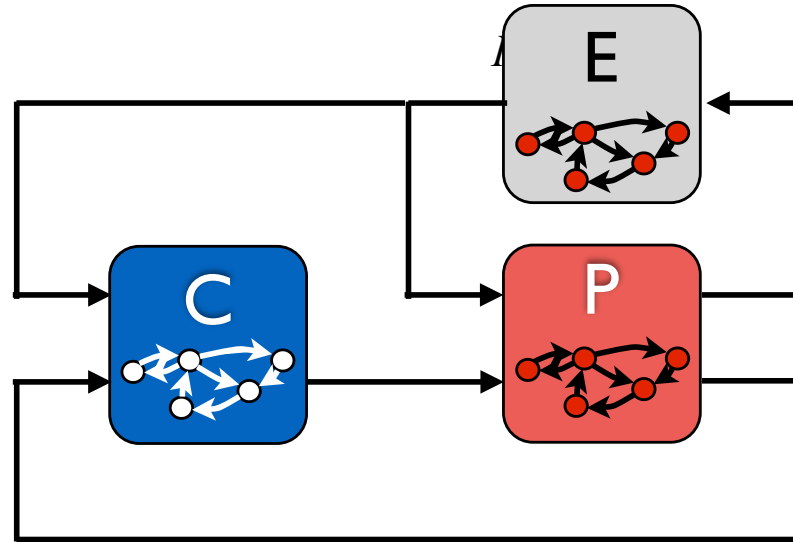
- Piterman, Pnueli and Sa'ar, 2006
- Assume environment fixes action before controller (breaks symmetry)
- For certain class of specifications, get complexity cubic in # of states (!)

$$(\phi_{\text{init}}^e \wedge \Box \phi_{\text{safe}}^e \wedge \Box \Diamond \phi_{\text{prog}}^e) \rightarrow (\phi_{\text{init}}^s \wedge \Box \phi_{\text{safe}}^s \wedge \Box \Diamond \phi_{\text{prog}}^s)$$

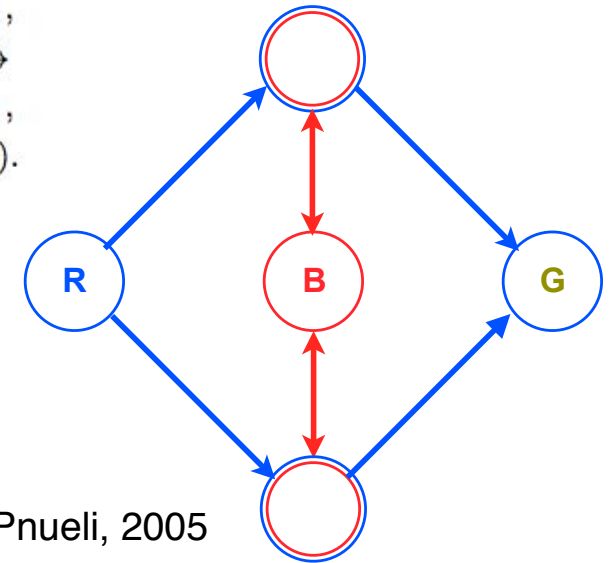
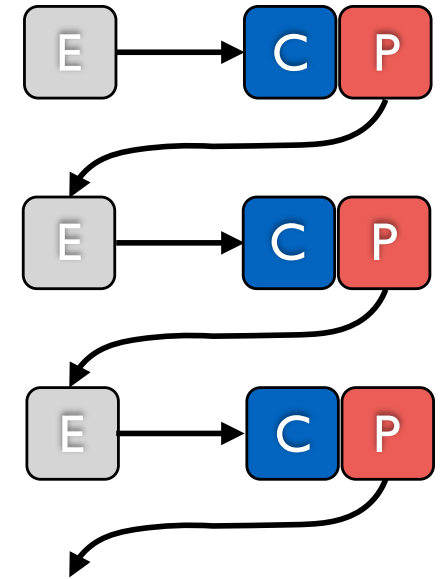
Environment assumption

System guarantee

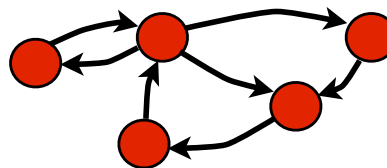
- GR(1) = general reactivity formula
- Assume/guarantee style specification



$$\begin{aligned} &\Box \{(\tilde{c} = 1 \wedge c = 0 \wedge (x_C < T_{c_{min}})) \rightarrow \\ &\quad (\bigcirc c = 0 \wedge \bigcirc x_C = x_C + \delta)\}, \\ &\Box \{(\tilde{c} = 1 \wedge c = 0 \wedge (x_C \geq T_{c_{min}})) \rightarrow \\ &\quad (\bigcirc c = 1 \vee \bigcirc x_C = x_C + \delta)\}, \\ &\Box (x_C \leq T_{c_{max}}). \end{aligned}$$



A. Pnueli, 2005



# Temporal Logic Planning (TuLiP) toolbox

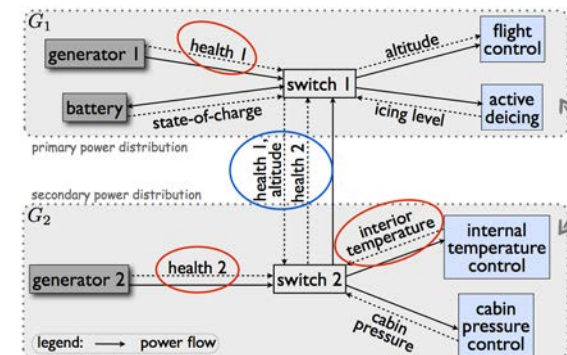
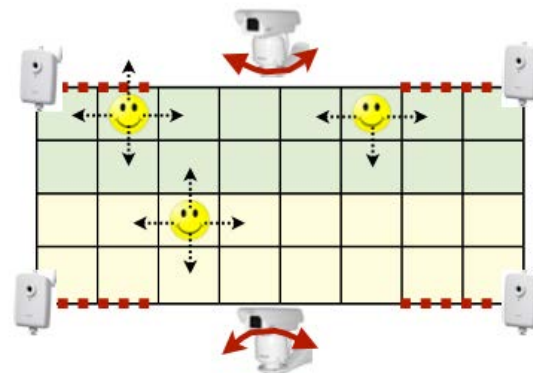
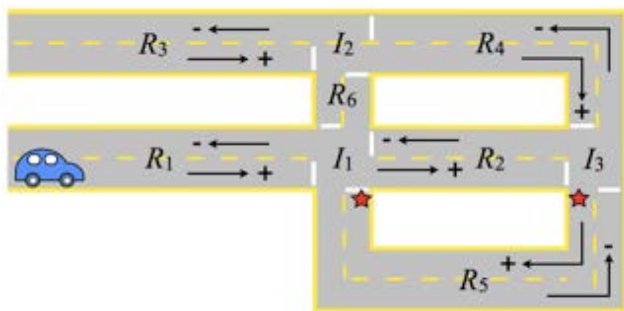
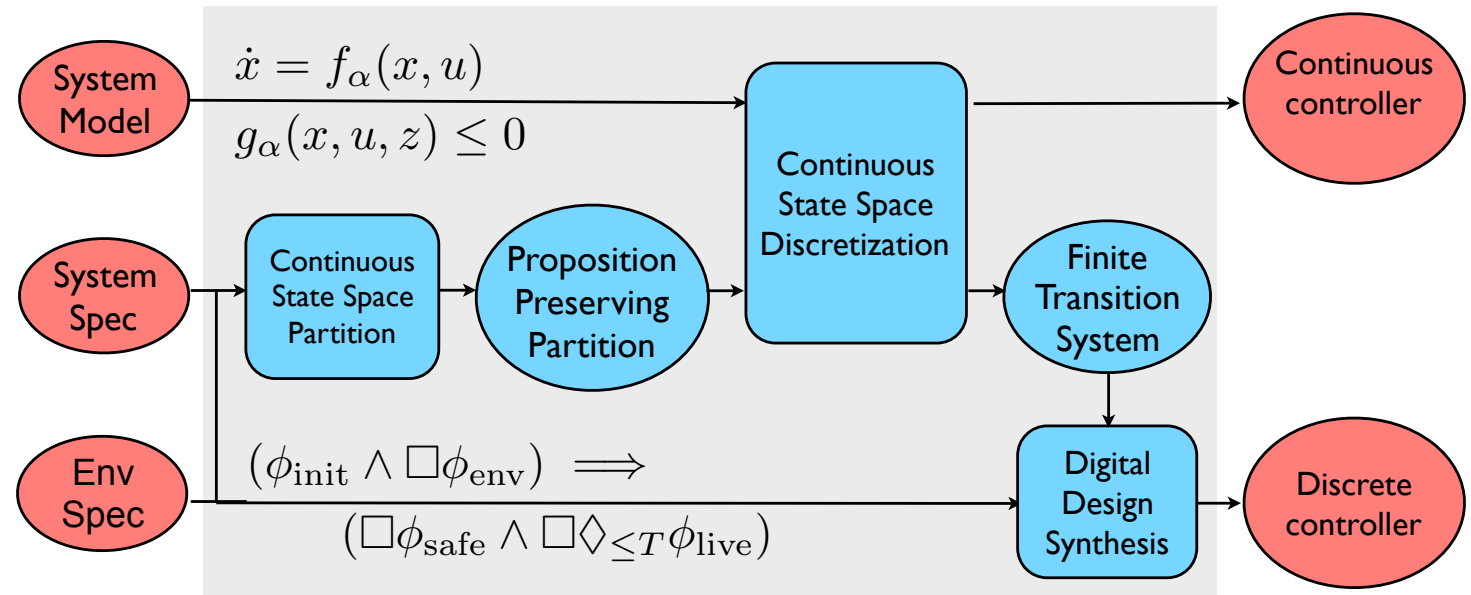
<http://tulip-control.org>

## Python Toolbox

- GR(1), LTL specs
- Nonlinear dynamics
- Supports discretization via MPT
- Control protocol designed w/ gr1c
- Receding horizon compatible

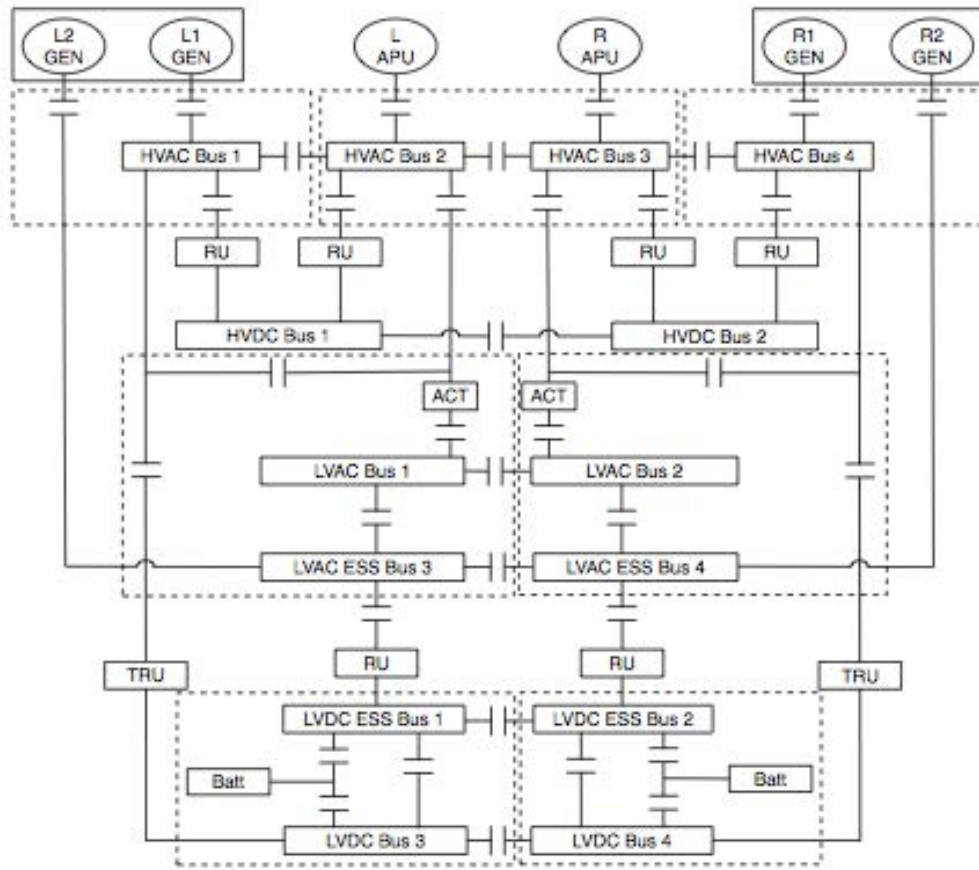
## Applications of TuLiP

- Autonomous vehicles - traffic planner (intersections and roads, with other vehicles)
- Distributed camera networks - cooperating cameras to track people in region
- Electric power transfer - fault-tolerant control of generator + switches + loads





# Example: Electric Power Systems



R. G. Michalko, "Electrical starting, generation, conversion and distribution system architecture for a more electric vehicle," US Patent 7,439,634 B2, Oct. 2008.

## REQUIREMENTS:

1. No AC bus shall be simultaneously powered by more than one AC source.
2. The aircraft electric power system shall provide power with the following characteristics: 115 +/- 5 V (amplitude) and 400 Hz (frequency) for AC loads and 28 +/- 2V for DC loads.
3. Buses shall be powered according to the priority tables.
4. AC buses shall not be unpowered for more than 50ms.
5. The overall system failure probability must be less than  $10^{-9}$  per flight hour.
6. Never lose more than one bus for any single failure.
7. Total load must be within the capacity of the generator

## Properties can be formulated in GR(1)

- Safety: supply power, avoid shorts/paralleling
- Progress: all loads eventually powered

## Verification

- Given properties + logic, ensure that specs are satisfied

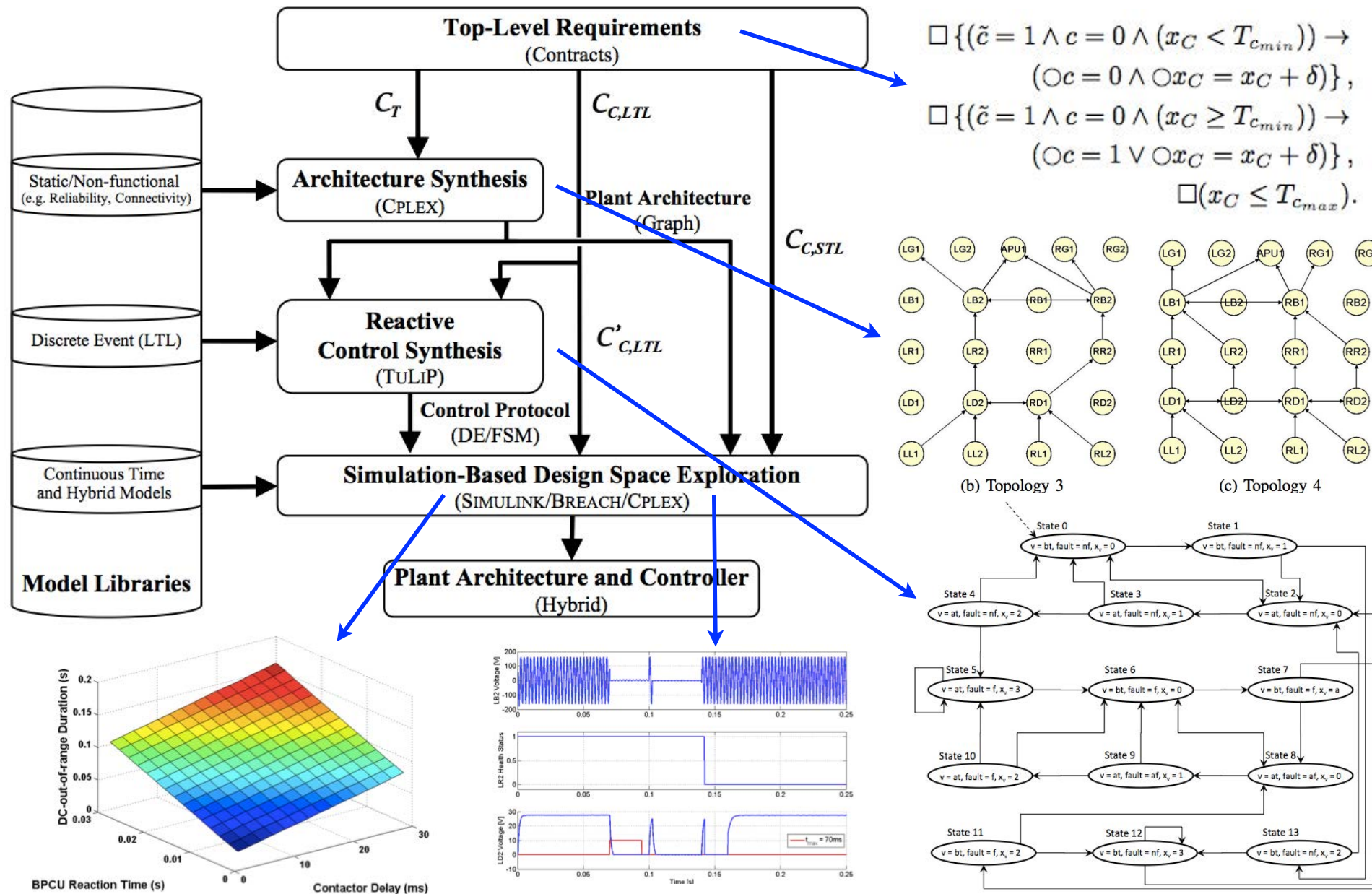
## Synthesis

- Given properties and topology + actuators, *synthesize* switching logic

## Component models/specifications:

1. Failure probabilities for contactors, generators, etc. (not much on failure modes)
2. Contactor closure times are between 15-25 ms and opening times are between 10-20 ms.

# EPS Design Space Exploration



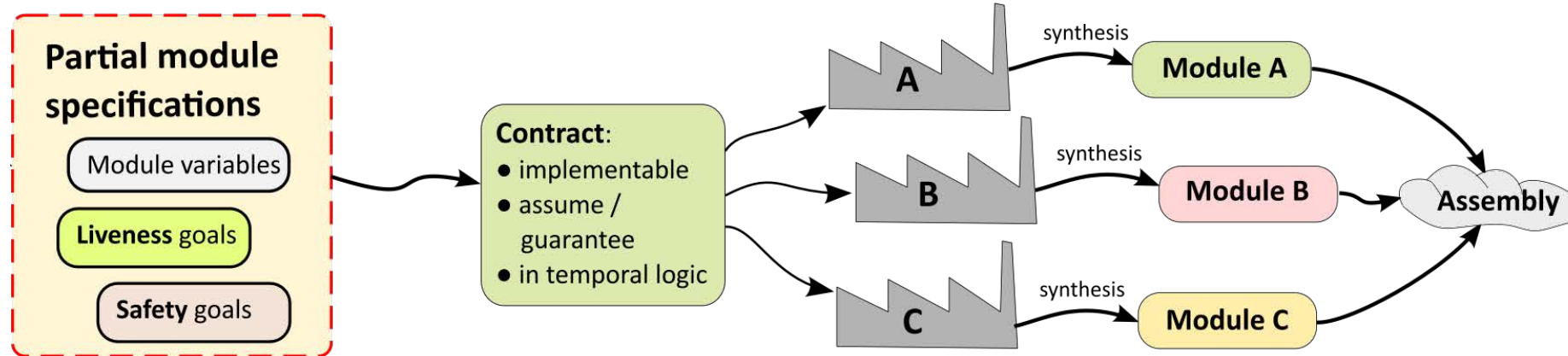
## Design workflow

- Formalize specs as a A/G contracts
- Synthesize possible EPS topologies
- Synthesize control logic, if possible
- Use more complex models to verify continuous time properties

## Applications

- Aircraft electric power systems
- Environmental control systems

# Structure of Specifications for a System



## Synthesis of contracts

- Given a set of (LTL) properties, *synthesize* GR(1) contracts for components
- Key component is amount of information that must be shared
  - Can minimize subject to constraints

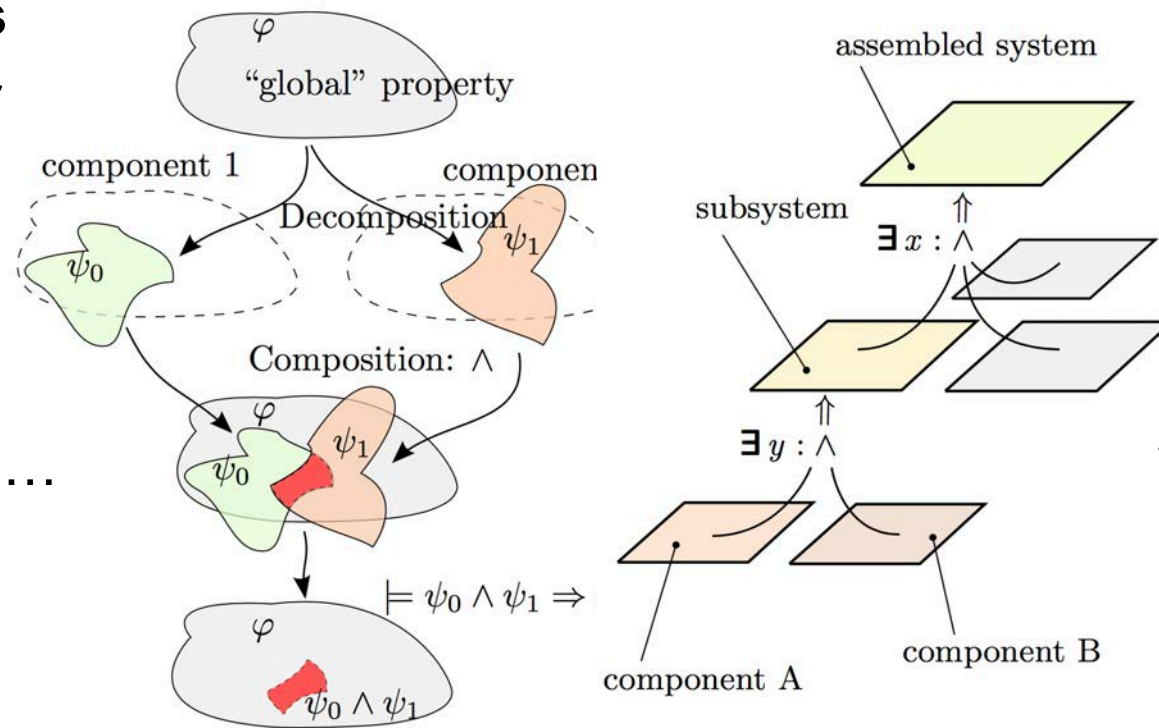
## Assume/guarantee contracts

- Assume: properties of other components in the system
- Guarantee: properties that will hold for my component

$$A_i \Rightarrow G_i$$

$$G_2 \wedge G_3 \Rightarrow A_1, G_1 \wedge G_3 \Rightarrow A_2, \dots$$

- Contracts can be horizontal (within a layer) or vertical (between two layers)

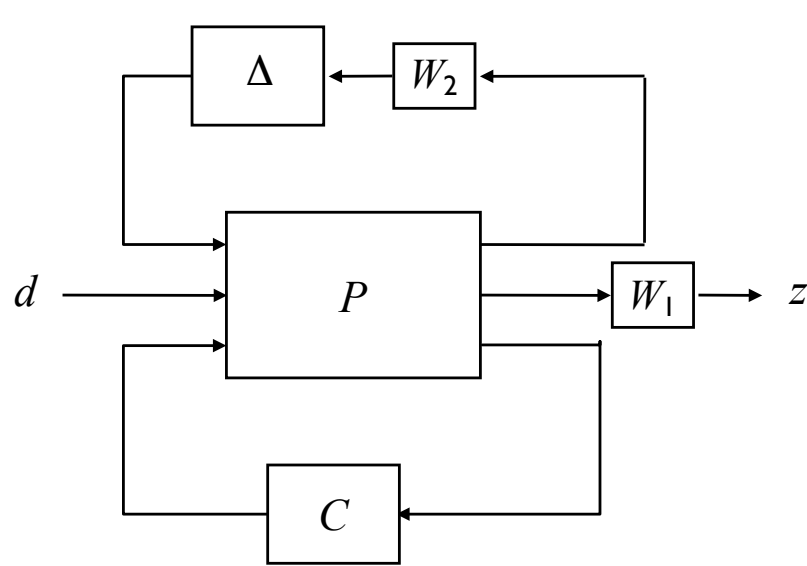


## Software (I. Filippidis)

- omega - synthesis of controllers/contracts
- dd - binary decision diagrams in Python

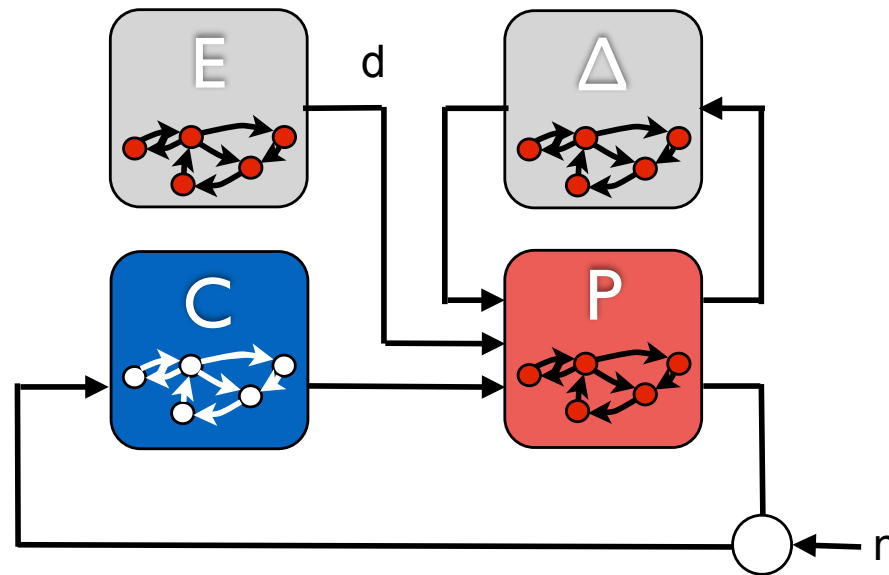


# Rapprochement Between Formal Methods and Control



$$\|z\|_2 \leq \gamma \|d\|_2 \text{ for all } \|\Delta\| \leq 1 \quad \square \phi_{\text{safe}}^e \wedge \square \diamond \phi_{\text{prog}}^e \rightarrow \square \phi_{\text{safe}}^s \wedge \square \diamond \leq T \phi_{\text{prog}}^s$$

Controlling cyberphysical systems requires solving *both* problems



Getting more rigorous about control of reactive systems

- Systems are too **complex** to be tested by trial and error
- Systems are too **safety-critical** to be tested by trial and error
- The way forward:
  - specify then synthesize
  - maintain layered structure
  - synthesis of contracts

